

Lecture 4: Representation Learning and Convolutional Networks

André Martins



Deep Structured Learning Course, Fall 2020

Announcements

Deadline for project proposal is today!

- You need to turn in a 1-page proposal (NeurIPS format) explaining:
 - The type of project: survey or practical
 - The problem you propose to solve / the topic you will address
 - For surveys: motivation and list of main papers you will cover
 - For practical projects:
 - Which method(s) you are going to use
 - Which evaluation metric and which data you are going to use

Homework 1 due next Wednesday.

Today's Roadmap

Today's lecture is about:

- Representation learning.
- Principal component analysis (PCA) and auto-encoders.
- Denoising auto-encoders.
- Distributed representations.
- Word embeddings and negative sampling.
- Multilingual and contextual word embeddings.
- Convolutional neural networks.
- Convolutions and max-pooling layers.

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

② Convolutional Neural Networks

③ Visualizing Representations

④ Conclusions

Representations

One of the greatest features of neural networks is their ability to **learn representations** $\phi(\mathbf{x})$

Contrast this with linear models, where features $\phi(\mathbf{x})$ are manually engineered

Representations are useful for several reasons:

- They can make our models more expressive and more accurate
- We may want to transfer representations from one task to another

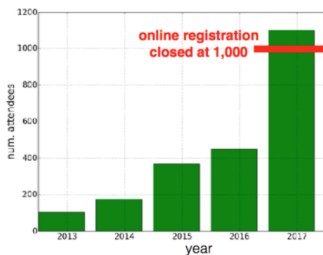
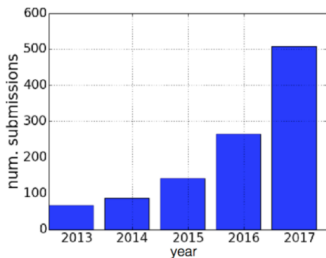
We talked about the first point when discussing the multi-layer perceptron

In this lecture, we'll focus on the second point.

Representation Learning

This is becoming a extremely popular topic!

Number of submissions at the “International Conference on Learning Representations” (ICLR):



Outline

1 Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

2 Convolutional Neural Networks

3 Visualizing Representations

4 Conclusions

Key Idea

Deeper neural networks learn **coarse-to-fine** representation layers.

Hierarchical Compositionality

Vision:

- pixels → edge → texton → motif → part → object → scene

Speech:

- audio sample → spectral band → formant → motif → phone → word

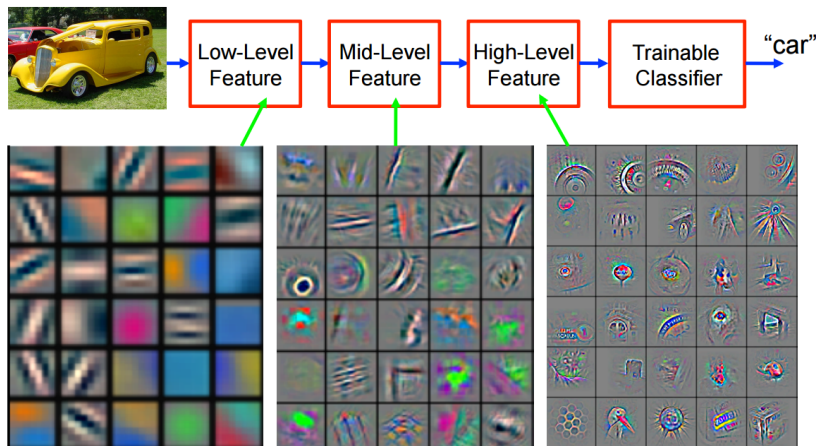
Text:

- character → word → phrase → sentence → story

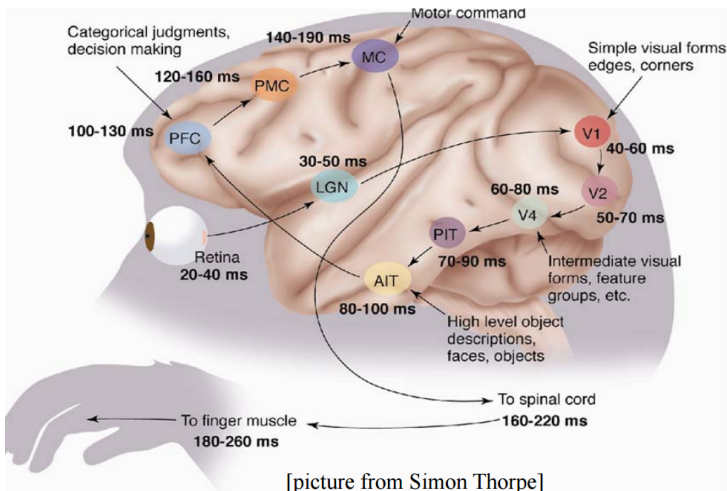
(Slide inspired by Marc'Aurelio Ranzato and Yann LeCun)

Hierarchical Compositionality

Feature visualization of convolutional net trained on ImageNet from Zeiler and Fergus (2013):



The Mammalian Visual Cortex is Hierarchical



(Slide inspired by Marc'Aurelio Ranzato and Yann LeCun)

What's in Each Layer

- Bottom level layers (closer to inputs) tend to learn **low-level representations** (corners, edges)
- Upper level layers (farther away from inputs) learn **more abstract representations** (shapes, forms, objects)

This holds for images, text, etc.

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

② Convolutional Neural Networks

③ Visualizing Representations

④ Conclusions

Distributed Representations (Hinton, 1984)

This is a central concept in neural networks.

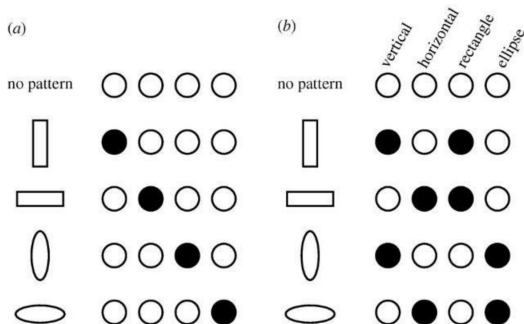
Key questions:

- How can a neural network be so effective representing objects when it has only a few hidden units (i.e. much fewer units than possible objects)?
- What is each hidden unit actually representing?
- How can a neural network generalize to objects that it has never seen before?

Local vs Distributed Representations

Consider two alternative representations:

- Local (one-hot) representations (one dimension per object)
- **Distributed representations** (one dimension per **property**)



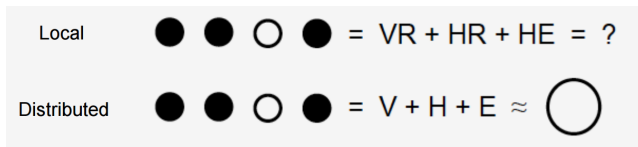
(Slide inspired by Moontae Lee and Dhruv Batra)

Distributed Representations

Key idea: no single neuron “encodes” everything; groups of neurons (e.g. in the same hidden layer) work together!

The Power of Distributed Representations

- Distributed representations are **more compact** (there can be $O(\exp N)$ objects combining N properties)
- They are also **more powerful**, as they can generalize to unseen objects in a meaningful way:



(Slide inspired by Moontae Lee and Dhruv Batra)

The Power of Distributed Representations

- For this to work, we need hidden units to capture diverse properties of the objects (i.e. we don't want all them to capture the same property)
- This is usually ensured by random initialization of the weights
- If we initialized all the units to the same weights, we would never break the symmetry!
- Side note: a neural network computes the same function if we permute the hidden units within the same layer (order doesn't matter, only diversity)

Next: how can we learn useful representations of objects from raw inputs only (i.e. no labels)?

Example: Unsupervised Pre-Training

Training deep networks (with many hidden layers) can be challenging
This has been a major difficulty with neural networks for a long time
Erhan et al. (2010): initialize hidden layers using **unsupervised learning**:

- Force network to **represent latent structure** of input distribution
- Encourage hidden layers to encode that structure
- This can be done with an **auto-encoder**!

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

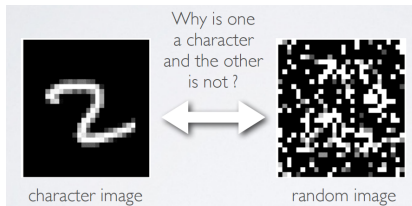
② Convolutional Neural Networks

③ Visualizing Representations

④ Conclusions

Data Manifold

Key idea: learn the manifold where the input objects live



(Image credit: Hugo Larochelle)

Learn representations that encode well points in that manifold

Auto-Encoders

An **auto-encoder** is a feed-forward neural network trained to reproduce its input at the output layer

Encoder:

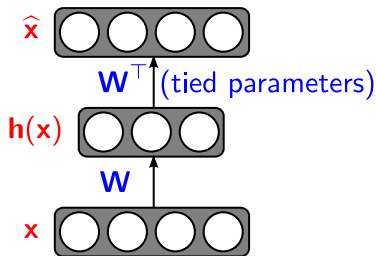
$$h(\mathbf{x}) = \mathbf{g}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Decoder:

$$\hat{\mathbf{x}} = \mathbf{W}^T h(\mathbf{x}) + \mathbf{c}$$

Loss function (for real-valued inputs):

$$L(\hat{\mathbf{x}}; \mathbf{x}) = \frac{1}{2} \|\hat{\mathbf{x}} - \mathbf{x}\|^2$$



The Simplest Auto-Encoder

What happens if the activation function g is linear?

The Simplest Auto-Encoder

What happens if the activation function g is linear?

Principal Component Analysis (PCA)!

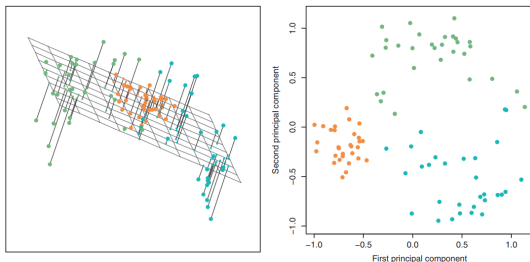


FIGURE 10.2. *Ninety observations simulated in three dimensions. Left: the first two principal component directions span the plane that best fits the data. It minimizes the sum of squared distances from each point to the plane. Right: the first two principal component score vectors give the coordinates of the projection of the 90 observations onto the plane. The variance in the plane is maximized.*

(From “An Introduction to Statistical Learning” by James, Witten, Hastie, Tibshirani)

Proof

Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be the data matrix (N examples, D features, $N > D$)

Assume $\mathbf{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (let's ignore the biases for simplicity and assume \mathbf{X} is centered)

We want to minimize $\|\mathbf{X} - \hat{\mathbf{X}}\|_F^2$, where $\hat{\mathbf{X}} = \mathbf{X}\mathbf{W}^\top\mathbf{W}$ is the **reconstruction matrix**, which by construction has rank K

From Eckart-Young theorem, the minimizer is a **truncated SVD** of \mathbf{X}^\top :

$$\hat{\mathbf{X}}^\top = \mathbf{U}_K \Sigma_K \mathbf{V}_K^\top,$$

where Σ_K is a diagonal matrix containing the top K singular values of \mathbf{X}^\top , and the columns of \mathbf{U}_K are the corresponding left singular vectors

The solution is $\mathbf{W} = \mathbf{U}_K^\top$, which gives as desired:

$$\hat{\mathbf{X}}^\top = \mathbf{W}^\top \mathbf{W} \mathbf{X}^\top = \mathbf{U}_K \mathbf{U}_K^\top \mathbf{U} \Sigma \mathbf{V}^\top = \mathbf{U}_K \Sigma_K \mathbf{V}_K^\top.$$

Auto-Encoders

PCA fits a **linear manifold** (affine space) to the data

By using **non-linear** activations, we obtain more sophisticated codes (i.e. representations).

We need some sort of regularization to:

- encourage a smooth representation (small perturbations of the input will lead to similar codes)
- avoid overfitting on the provided inputs

Some Variants of Auto-Encoders

- **Sparse auto-encoders:** use many hidden units, but add a ℓ_1 regularization term to encourage **sparse representations** of the input
- **Denoising auto-encoders:** regularize by adding noise to the input; the goal is to learn a **smooth representation function** that allows to output the denoised input (inspired by image denoising)
- **Stacked auto-encoders:** stack several auto-encoders on top of each other
- **Variational auto-encoders:** a generative probabilistic model that minimizes a variational bound (this will be covered in another lecture!)

Regularized Auto-Encoders

To regularize auto-encoders, we may add a **regularization term** to the loss

The goal is then to minimize $L(\hat{\mathbf{x}}; \mathbf{x}) + \Omega(\mathbf{h}, \mathbf{x})$

For example:

- regularizing the code $\Omega(\mathbf{h}, \mathbf{x}) = \lambda \|\mathbf{h}\|^2$
- regularizing the derivatives $\Omega(\mathbf{h}, \mathbf{x}) = \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2$

The encoder and decoder parameters may be shared or not.

Sparse Auto-Encoders

Most auto-encoders learn low-dimensional codes, e.g., they reduce input dimensionality (bottleneck shape $K < D$).

But one exception are **sparse auto-encoders**:

- Sparse auto-encoders incorporate a sparsity penalty $\Omega(\mathbf{h})$ on the code layer, e.g., $\Omega(\mathbf{h}) = \lambda \|\mathbf{h}\|_1$
- Typically the number of hidden units is large, e.g., larger than the input dimension
- The sparsity penalty encourages sparse codes, where most hidden units are inactive.

Stochastic Auto-Encoders

In this case, the encoder and decoder are not deterministic functions, but involve some noise injection

We have a distribution $p_{\text{encoder}}(\mathbf{h} | \mathbf{x})$ for the encoder and a distribution $p_{\text{decoder}}(\mathbf{x} | \mathbf{h})$ for the decoder

The auto-encoder can be trained to minimize

$$-\log p_{\text{decoder}}(\mathbf{x} | \mathbf{h}).$$

Denosing Auto-Encoders

- Use a perturbed version of the input, $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{n}$, where \mathbf{n} is random noise (e.g. Gaussian noise $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$)
- Instead of minimizing $\frac{1}{2} \|\hat{\mathbf{x}} - \mathbf{x}\|^2$, minimize $\frac{1}{2} \|\hat{\mathbf{x}} - \tilde{\mathbf{x}}\|^2$
- This is a form of implicit regularization that ensures **smoothness**: it forces the system to represent well not only the data points, but also their perturbations

Denosing Auto-Encoders

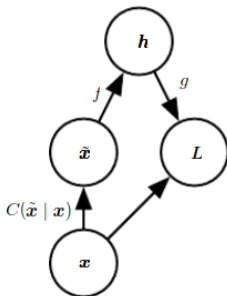


Figure 14.3: The computational graph of the cost function for a denoising autoencoder, which is trained to reconstruct the clean data point \mathbf{x} from its corrupted version $\tilde{\mathbf{x}}$. This is accomplished by minimizing the loss $L = -\log p_{\text{decoder}}(\mathbf{x} | \mathbf{h} = f(\tilde{\mathbf{x}}))$, where $\tilde{\mathbf{x}}$ is a corrupted version of the data example \mathbf{x} , obtained through a given corruption process $C(\tilde{\mathbf{x}} | \mathbf{x})$. Typically the distribution p_{decoder} is a factorial distribution whose mean parameters are emitted by a feedforward network g .

(From Goodfellow et al.'s book.)

Denosing Auto-Encoders

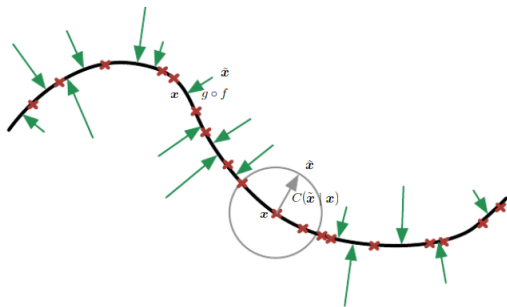


Figure 14.4: A denoising auto-encoder is trained to map a corrupted data point $\tilde{\mathbf{x}}$ back to the original data point \mathbf{x} . We illustrate training examples \mathbf{x} as red crosses lying near a low-dimensional manifold, illustrated with the bold black line. We illustrate the corruption process $C(\tilde{\mathbf{x}} | \mathbf{x})$ with a gray circle of equiprobable corruptions. A gray arrow demonstrates how one training example is transformed into one sample from this corruption process. When the denoising auto-encoder is trained to minimize the average of squared errors $\|g(f(\tilde{\mathbf{x}})) - \mathbf{x}\|^2$, the reconstruction $g(f(\tilde{\mathbf{x}}))$ estimates $\mathbb{E}_{\mathbf{x}, \tilde{\mathbf{x}} \sim p_{\text{data}}(\mathbf{x}) C(\tilde{\mathbf{x}} | \mathbf{x})}[\mathbf{x} | \tilde{\mathbf{x}}]$. The vector $g(f(\tilde{\mathbf{x}})) - \tilde{\mathbf{x}}$ points approximately toward the nearest point on the manifold, since $g(f(\tilde{\mathbf{x}}))$ estimates the center of mass of the clean points \mathbf{x} that could have given rise to $\tilde{\mathbf{x}}$. The auto-encoder thus learns a vector field $g(f(\mathbf{x})) - \mathbf{x}$ indicated by the green arrows. This vector field estimates the score $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ up to a multiplicative factor that is the average root mean square reconstruction error.

Why Do We Use Auto-Encoders?

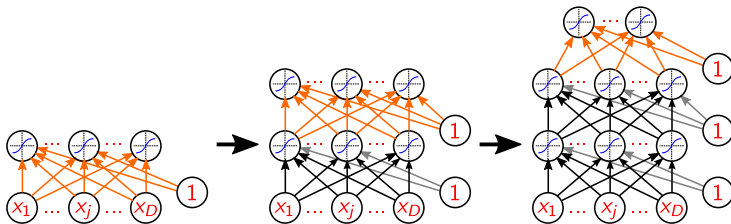
Historically, training deep neural networks was hard

One of the initial successful uses of auto-encoders was for **unsupervised pre-training** (Erhan et al., 2010).

Unsupervised Pre-Training

A greedy, layer-wise procedure:

- train one layer at a time, from first to last, with unsupervised criterion (e.g. an auto-encoder)
- fix the parameters of previous hidden layers
- previous layers viewed as feature extraction



Pre-training initializes the parameters in a region such that the near local optima overfit less the data.

Fine-Tuning

Once all layers are pre-trained:

- add output layer
- train the whole network using supervised learning

Supervised learning is performed as in a regular feed-forward network:

- forward propagation, backpropagation and update
- all parameters are “tuned” for the supervised task at hand
- representation is adjusted to be more discriminative

Other Applications of Auto-Encoders

- Dimensionality reduction
- Information retrieval and semantic hashing (via binarizing the codes)
- Conversion of discrete inputs to low-dimensional continuous space

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

② Convolutional Neural Networks

③ Visualizing Representations

④ Conclusions

Word Representations

We'll focus now on recent methods for learning representations of **words in natural language**

Also called **word embeddings**

This has been an extremely successful application of representation learning

It's still a very active area of research!

Distributional Similarity

Key idea: represent a word by means of its neighbors

- “*You shall know a word by the company it keeps*” (J. R. Firth, 1957)
- One of the most successful ideas of modern statistical NLP!

For example:

- Adjectives are normally surrounded by nouns
- Words like *book, newspaper, article*, are commonly surrounded by *reading, read, writes*, but not by *flying, eating, sleeping*

We have seen an instance of this principle when we discussed **Brown clustering**.

Recap: Brown clustering

An example of (non-neural) **word representation learning**

It obtains **discrete word representations** (binary vectors) via class-based language models and unsupervised hierarchical clustering

It was extremely popular in NLP before the age of neural networks!

Today we'll look at ways of learning **continuous word representations**.

Examples of Brown clusters (from Twitter data)

Path	Terms
001010110	never neva nvr gladly nevr #never nevrr nver nevrrr nevaa
001010111	ever eva evar evr evrrr evrr evrrrr evah evrrrrr evrrrrr
01000010	does duz doess does sayeth doez doesss d0es deos

Path	Terms
0100	Monday
010100	Sunday
010101	Friday
0101100	Thursday
01011010	Saturday

(from http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html)

Word Embeddings

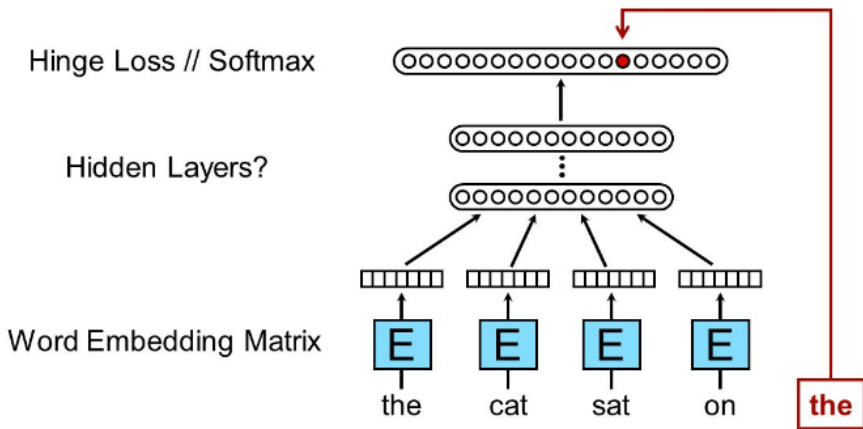
How do we obtain lower dimensional vector representations of words?

Two possible methods:

- Factorization of a co-occurrence word/context matrix (latent semantic analysis, etc.)
- Directly learn low-dimensional vectors by training a network to *predict* the context of a given word

We'll focus on the latter, incarnated in the **word2vec** toolkit (Mikolov et al., 2013), which follows previous ideas of Bengio et al. (2003) and Collobert et al. (2011).

Neural Language Model (Bengio et al., 2003)



(Image credits: Quoc Le)

Neural Language Model (Bengio et al., 2003)

- Each word is associated with a continuous vector (a **word embedding**)
- Given the **context** (previous K words), predict the next word
- This is done by concatenating the word embeddings in the context window, then propagating them through a feedforward neural network
- The output layer is a gigantic softmax that assigns a probability value to each word in the vocabulary

Variants of this model achieved better accuracy than smoothed K -th order Markov models

As a by-product: **word embeddings!**

The embedding matrix is a lookup table that assigns a continuous vector to every word in the vocabulary.

Neural Language Model

In this class, we are not concerned with language modeling (the actual task), but rather about the **quality of the embeddings** (the representations we learn for that task).

Some Insights

If we don't care about language modeling as a task:

- 1 We don't need to have a “left-to-right model” where we try to predict the next word given the context
- 2 We don't need to predict the probability of every word, we might just make sure that the true word is more likely than a random word

These insights underlie the word2vec model of Mikolov et al. (2013).

Word2Vec (Mikolov et al., 2013)

Considers a context window **around** each word in the sentence.

Word2vec comes with two variants:

- **Skip-gram**: predict surrounding context words in a window of length m of every word
- **Continuous bag-of-words (CBOW)**: predict the central word from the context

We'll focus on the skip-gram model (more widely used).

Skip-Gram

Goal: maximize the log probability of any context word given the current center word:

$$J(\Theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p_{\Theta}(x_{t+j} | x_t)$$

There are two sets of parameters $\Theta = (\mathbf{u}, \mathbf{v})$:

- Embeddings \mathbf{u}_o for each word o appearing as the center word
- Embeddings \mathbf{v}_c for each word c appearing in the context of another word

Define a **log-bilinear model**: $p_{\Theta}(x_{t+j} = c | x_t = o) \propto \exp(\mathbf{u}_o \cdot \mathbf{v}_c)$

Every word gets two vectors!

In the end, we use the \mathbf{u} vectors as the word embeddings and discard the \mathbf{v} vectors

The Large Vocabulary Problem

Recall that we have

$$p_{\Theta}(x_{t+j} = c \mid x_t = o) = \frac{\exp(\mathbf{u}_o \cdot \mathbf{v}_c)}{\sum_c' \exp(\mathbf{u}_o \cdot \mathbf{v}_c')}$$

This objective requires a softmax over the entire vocabulary

Unfortunately, with large vocabularies this leads to very slow training :(

Workarounds:

- Stochastic sampling
- Noise contrastive estimation
- Negative sampling

More details in these notes: <https://arxiv.org/pdf/1410.8251.pdf>

We'll focus on **negative sampling**.

Negative Sampling

Key idea:

- replace the gigantic softmax by **binary logistic regressions** for a true pair (center word and word in its context window) and a couple of random pairs (the center word with a random word):

$$J_t(\Theta) = \log \sigma(\mathbf{u}_o \cdot \mathbf{v}_c) + \sum_{i=1}^k \log \sigma(-\mathbf{u}_o \cdot \mathbf{v}_{j_i}), \quad j_i \sim P(x)$$

- Several strategies for the sampling distribution $P(x)$ (uniform, unigram frequency, etc.)

Negative sampling is a simple form of unsupervised pre-training.

Linear Relationships

- These representations are very good at encoding dimensions of similarity!
- **Word analogies** can be solved quite well just by doing vector subtraction in the embedding space
- Syntactically:

$$\mathbf{x}_{\text{apple}} - \mathbf{x}_{\text{apples}} \approx \mathbf{x}_{\text{car}} - \mathbf{x}_{\text{cars}} \approx \mathbf{x}_{\text{family}} - \mathbf{x}_{\text{families}}$$

- Semantically:

$$\begin{aligned}\mathbf{x}_{\text{shirt}} - \mathbf{x}_{\text{clothing}} &\approx \mathbf{x}_{\text{chair}} - \mathbf{x}_{\text{furniture}} \\ \mathbf{x}_{\text{king}} - \mathbf{x}_{\text{man}} &\approx \mathbf{x}_{\text{queen}} - \mathbf{x}_{\text{woman}}\end{aligned}$$

Visualization

Typical word embedding dimensions are on the hundreds (e.g. 300)

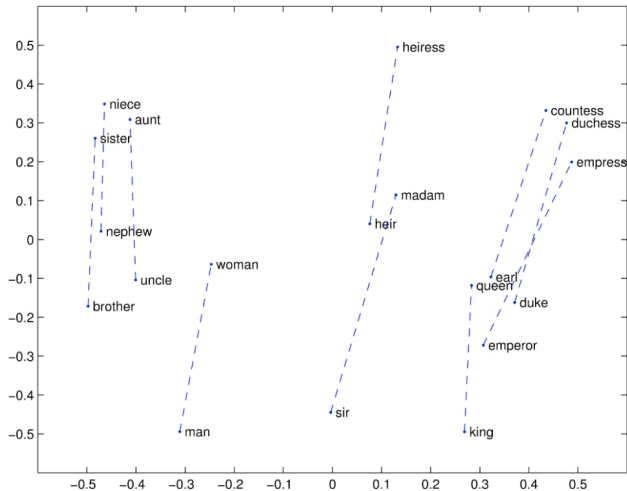
How can we visualize these embeddings?

Simple way: project them in 2D with something like PCA!

Most used: t -distributed stochastic neighbor embedding (t-SNE, Maaten and Hinton 2008)

<https://lvdmaaten.github.io/tsne>

Word Analogies (Mikolov et al., 2013)



(Slide credit to Richard Socher)

Other Methods for Obtaining Word Embeddings

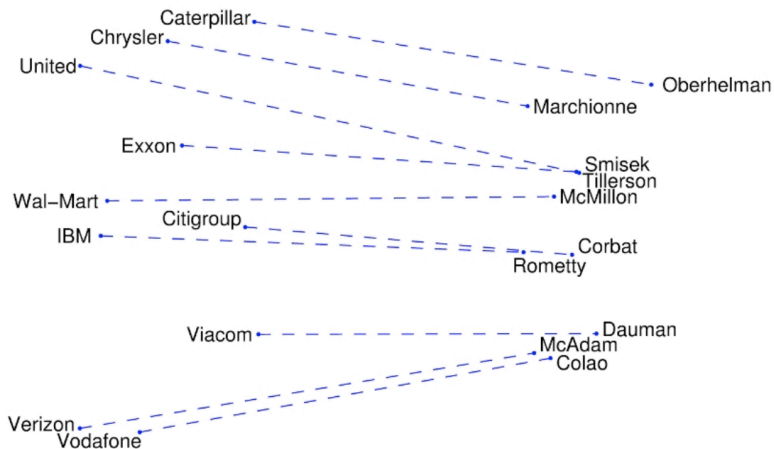
GloVe: Global Vectors for Word Representation (Pennington et al., 2014)

- <https://nlp.stanford.edu/projects/glove>
- Training is performed on aggregated global word-word co-occurrence statistics from a corpus

fastText (Bojanowski et al., 2016): embeds also character n -grams for generating embeddings for out-of-vocabulary words

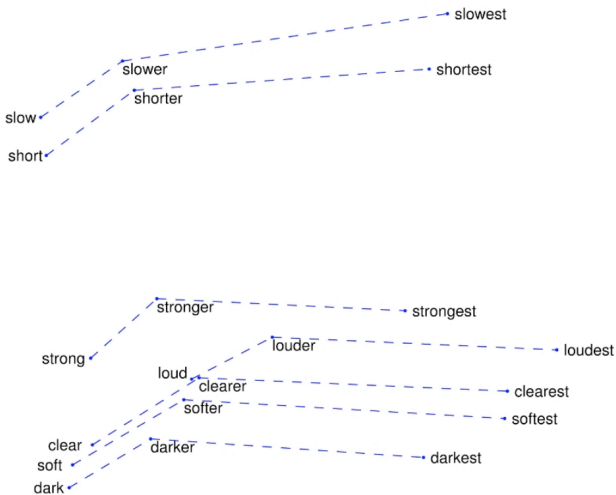
- <https://fasttext.cc> (from FAIR)
- open-source, free, lightweight library that allows users to learn text representations and text classifiers
- contains multi-lingual word vectors for 157 different languages

GloVe Visualizations: Company → CEO



(Slide credit to Richard Socher)

GloVe Visualizations: Superlatives



(Slide credit to Richard Socher)

Word Embeddings: Some Open Problems

- Can we have word embeddings for multiple languages in the same space?
- How to capture polysemy?
- These word embeddings are static, can we compute embeddings on-the-fly depending on the context?

Cross-Lingual Word Embeddings

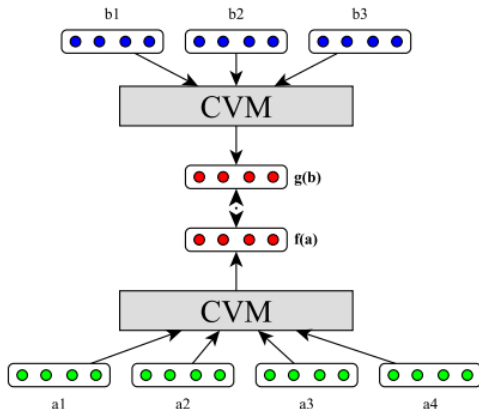


Figure 1: Model with parallel input sentences a and b . The model minimises the distance between the sentence level encoding of the bitext. Any composition functions (CVM) can be used to generate the compositional sentence level representations.

Cross-Lingual Word Embeddings

Key idea:

- use a corpus of parallel sentences in two languages
- define a composition function to obtain a sentence representation given word embeddings
- apply a loss function that encourages the sentence representations in the two languages to be similar
- negative sampling works here too: true pair vs fake pair.

Other approaches:

- Define a bilingual dictionary and apply canonical correlation analysis (Faruqui and Dyer, 2014)
- Task-specific embeddings with convex optimization (Ferreira et al., 2016)
- Learn the two embeddings separately, and then apply a linear transformation to put them in a shared space (Artetxe et al., 2017)
- Adversarial training (Lample et al., 2018)

This is a very active area of research!

Contextual Embeddings

Words can have different meanings, depending on which context they appear in.

In 2018, a model called ELMo learned **context-dependent embeddings** and achieved impressive results on 6 NLP downstream tasks (Peters et al., 2018)

Key idea:

- Pre-train a BiLSTM language model on a large dataset (we'll see in a later class what this is)
- Save **all** the encoder parameters at all layers, not only the embeddings
- Then, for your downstream task, tune a scalar parameter for each layer, and pass **the entire sentence** through this encoder.

BERT, GPT-2

Some time later, a Transformer-based model (BERT) achieved even better performance:

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

Huge improvements in multiple NLP tasks!

(Trained on 64 TPU chips!!)

Other related models include GPT-2, XLNet, KERMIT, etc.

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

② Convolutional Neural Networks

③ Visualizing Representations

④ Conclusions

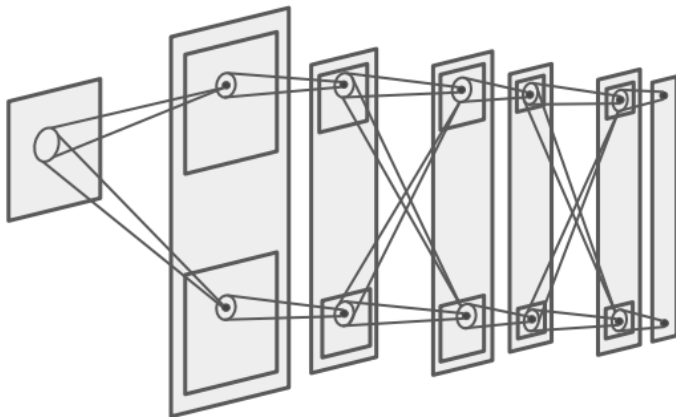
Convolutional Neural Networks

Convolutional Neural Networks are neural networks with **specialized connectivity structure**

Roadmap:

- Parameter Tying
- 2D Convolutional Nets for Object Recognition
- Pooling
- ImageNet, AlexNet, GoogLeNet
- 1D Convolutional Nets in NLP

Neocognitron (Fukushima and Miyake, 1982)



(Credits: Fei-Fei Li, Johnson, Yeung)

- “Sandwich” architecture (alternating simple cells with modifiable parameters and complex cells which perform pooling)

Neocognitron (Fukushima and Miyake, 1982)

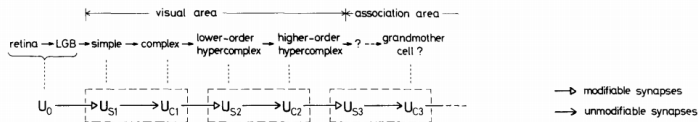


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

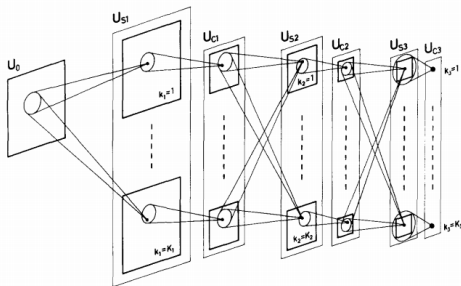
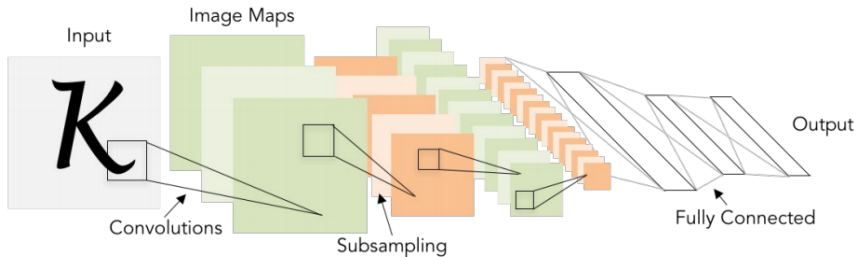


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

- Inspired by the multi-stage hierarchy model of the visual nervous system (Hubel and Wiesel, 1965)

ConvNet (LeNet-5) (LeCun et al., 1998)



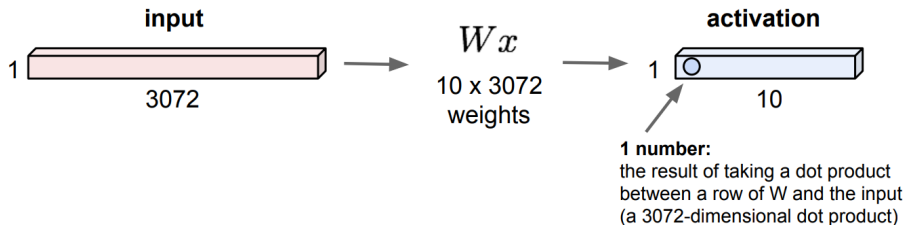
Convolutional Networks

... but what is a **convolutional layer** after all?

Let's compare it with a fully connected layer (as in a standard feedforward neural network).

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



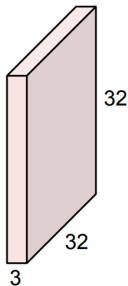
(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

Don't stretch: preserve the spatial structure!

Convolution Layer

32x32x3 image



Filters always extend the full depth of the input volume

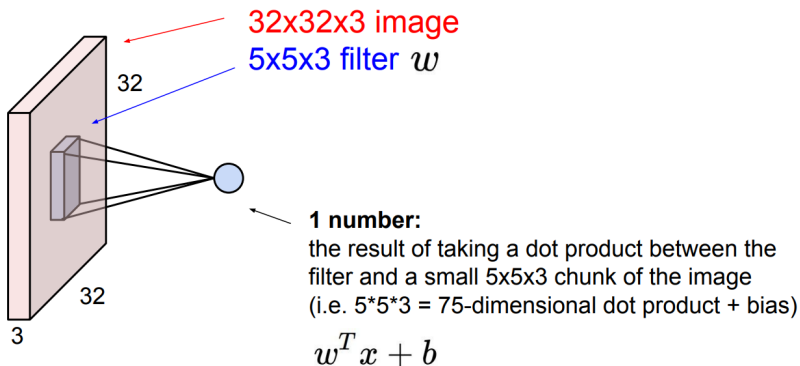
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

(Credits: Fei-Fei Li, Johnson, Yeung)

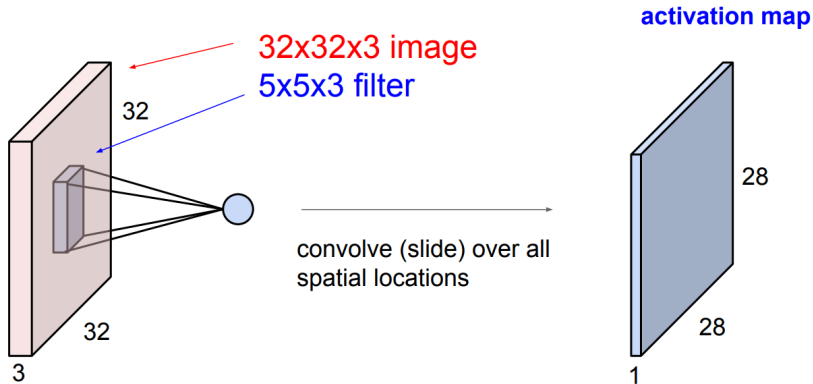
Convolutional Layer



(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

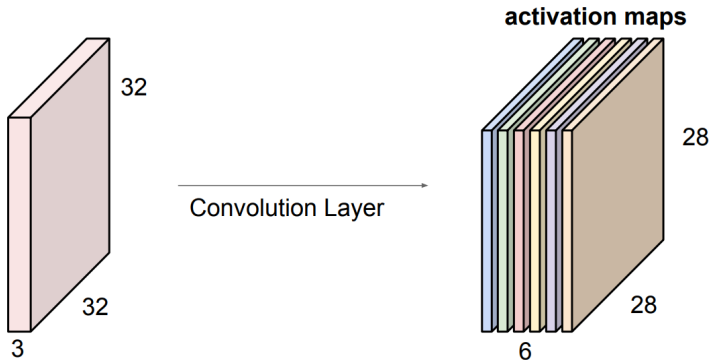
Apply the same filter to all spatial locations (28x28 times):



(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

- For example, if we have 6 5×5 filters, we get 6 separate activation maps:



(Credits: Fei-Fei Li, Johnson, Yeung)

- We stack these up to get a “new image” of size $28 \times 28 \times 6$!

Image Size, Filter Size, Stride, Channels

Stride is the shift in pixels between two consecutive windows

So far we have considered a **stride** of 1

The number of **channels** is the number of filters we consider in each layer

Given an $N \times N \times D$ image, $F \times F \times D$ filters, K channels, and stride S , the resulting output will be of size $M \times M \times K$, where

$$M = (N - F)/S + 1$$

For example:

- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 1$ results in an $28 \times 28 \times 6$ output
- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 3$ results in an $10 \times 10 \times 6$ output

In practice: common to pad the border with zeros

Common pad size is $(F - 1)/2$, which preserves size spatially.

The Brain View

The number of channels corresponds to the number of **neurons**

The filter size is also called the **receptive field** of each neuron.

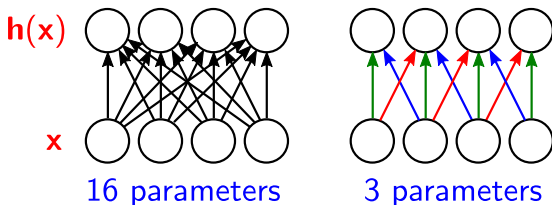
Convolutions and Parameter Tying

Why do we call this “convolutional”?

The **convolution** of a signal and a filter is:

$$h[t] = (x * w)[t] = \sum_{a=-\infty}^{\infty} x[a]w[t - a].$$

The basic idea of conv nets is the combination of **sparse connectivity** and **parameter tying**.



Convolutions and Parameter Tying

Leads to **translation equivariance**

Why do we want to tie (share) parameters?

- Reduce the number of parameters to be learned
- Deal with arbitrary long, variable-length, sequences

Can be done in 1D as well (common in textual data)!

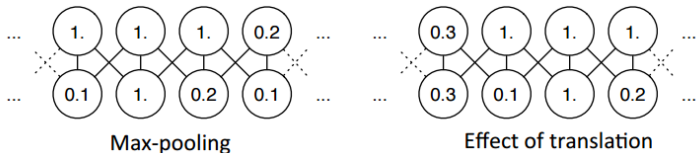
Convolutions and Pooling

The second component of conv nets is **pooling**

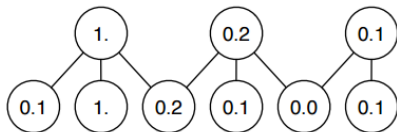
Common conv nets alternate convolutional layers and pooling layers.

Pooling Layers

- Aggregate to achieve local invariance:



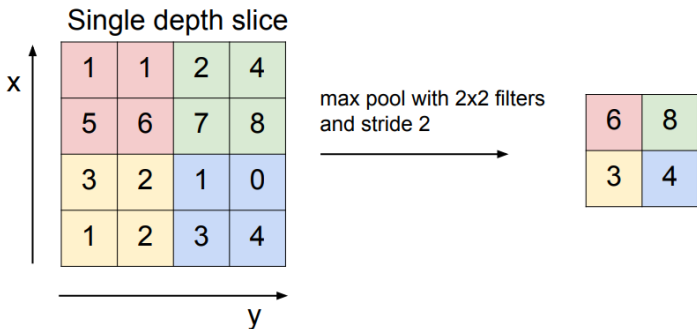
- Subsampling to reduce temporal/spacial scale and computation:



(Slide credit to Yoshua Bengio)

Pooling Layer

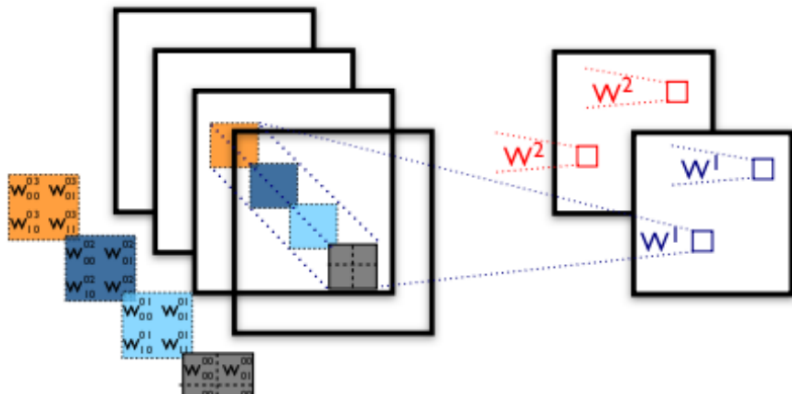
- Makes the representations smaller and more manageable
- Operates over each activation map (each channel) independently
- Max-pooling:



(Credits: Fei-Fei Li, Johnson, Yeung)

Multiple Convolutions: Feature Maps

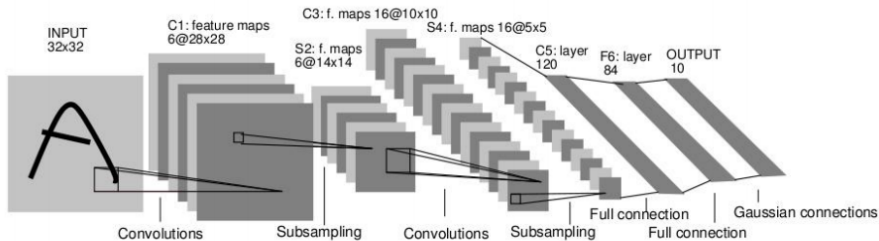
- Different filter weights for each channel, but keeping spatial invariance:



(Slide credit to Yoshua Bengio)

2D Convolutional Nets (LeCun et al., 1989)

- Inspired by “Neocognitron” (Fukushima, 1980)
- 2D Convolutions: the same filter (e.g. 3×3) is applied to each location of the image
- The filter weights are learned (as tied parameters)
- Multiple filters
- Alternates convolutional and pooling layers.



ConvNet Successes: MNIST



Handwritten text/digits:

- MNIST (0.35% error (Ciresan et al., 2011b))
- Arabic and Chinese (Ciresan et al., 2011a)

ConvNet Successes: CIFAR-10, Traffic Signs



Simpler recognition benchmarks:

- CIFAR-10 (9.3% error (Wan et al., 2013))
- Traffic signs: 0.56% error vs 1.16% for humans (Cireşan et al., 2011)

But less good at more complex datasets, e.g. Caltech-101/256 (few training examples).

ImageNet Dataset

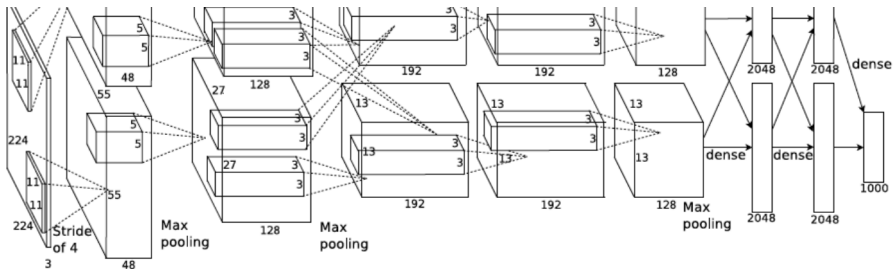
- 14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk



(Slide credit to Rob Fergus)

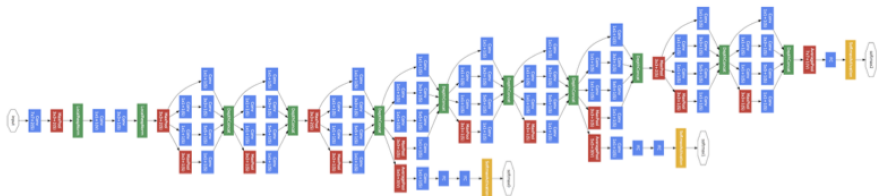
AlexNet (Krizhevsky et al., 2012)

- 54M parameters; 8 layers (5 conv, 3 fully-connected)
- Trained on 1.4M ImageNet images
- Trained on 2 GPUs for a week (50x speed-up over CPU)
- Dropout regularization
- Test error: 16.4% (second best team was 26.2%)



GoogLeNet (Szegedy et al., 2015)

- GoogLeNet inception module: very deep convolutional network, fewer (5M) parameters



Convolutional Nets in NLP

So far, we talked mostly about images.

Are conv nets also used in NLP? Not as much, but...

Quoting Yoav Goldberg in the Representation Learning Workshop in ACL 2018:

“NLP’s ImageNet moment has arrived.”

(Not referring to conv nets in particular, but to big neural architectures.)

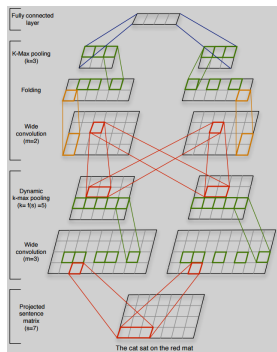
Convolutional Nets in NLP

- 1D convolutions
- Filters are applied to local windows around each word
- For word embeddings $\mathbf{x}_1, \dots, \mathbf{x}_L$, the filter response for word i is:

$$\mathbf{h}_i = \mathbf{g}(\mathbf{W}[\mathbf{x}_{i-h} \oplus \dots \oplus \mathbf{x}_i \oplus \dots \oplus \mathbf{x}_{i+h}] + \mathbf{b}),$$

where \oplus denotes vector concatenation and \mathbf{W} are shared parameters

- Can pad left and right with special symbols if necessary.



Kalchbrenner et al. (2014)

Variable Input Length

Most computation in conv nets can be done in parallel

GPUs can leverage this and achieve great speed-ups!

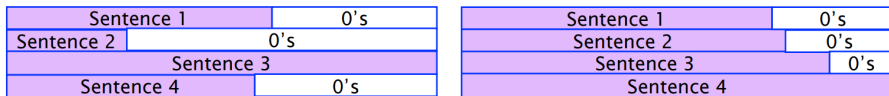
But unlike images which have fixed size, sentences have different lengths (number of words), which makes batching a bit trickier!

Mini-Batching, Padding, and Masking

Mini-batching is necessary to speed up training in GPUs

But how to cope with different input sizes (e.g. different sentence lengths)?

Solution: Minimize waste by sorting by sentence length before forming mini-batches, then **padding**:



(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

Masking needs to be used to make sure the padded symbols are not affecting the results.

Beyond Convolutions

Other architectures have been proposed which offer alternatives to convolutions

For example: **transformer networks**, which stack multi-head attention layers

This is somewhat similar to “dynamic convolutions”

We’ll cover this in another lecture.

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

② Convolutional Neural Networks

③ Visualizing Representations

④ Conclusions

What Representations Are We Learning?

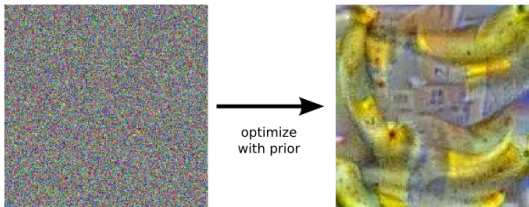
Which neurons fire for recognizing a particular object?

What parts of the network are activated?

To understand this, we need a way of **visualizing** what's happening inside the network.

Visualization

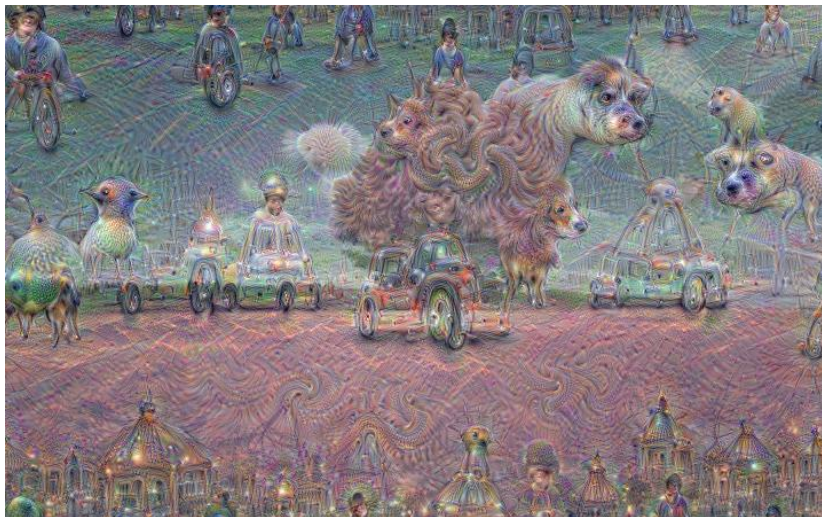
- **Idea:** Optimize input to maximize particular output
- Depends on the initialization
- **Google DeepDream**, maximizing “banana” output:



(from <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>)

- Can also specify a particular layer and tune the input to maximize the layer’s activations—useful to see what kind of features each layer is representing
- Specifying a higher layer produces more complex representations...

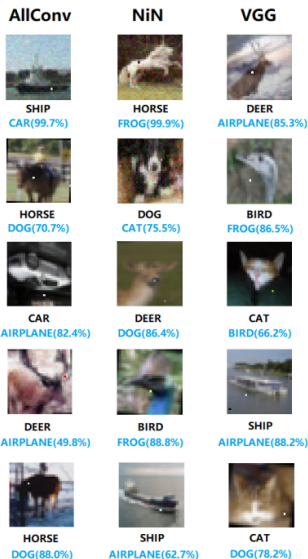
Google DeepDream



(from <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>)

Adversarial Attacks

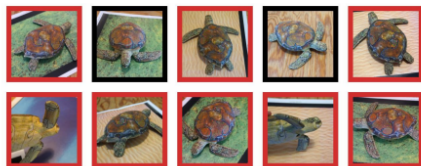
- How can we perturb an input slightly to fool a classifier?
- For example: **1-pixel attacks**
- **Glass-box model**: assumes access to the model
- Backpropagate to the inputs to find pixels which maximize the gradient
- There's also work for **black-box** adversarial attacks (don't have access to the model, but can query it).



◀ (Credits: Su, Vargas, Sakurai (2018)) 🔍 ↻

Even Worse: Perturb Object, Not Image

- Print the model of a turtle in a 3D printer.
- Perturbing the texture fools the model into thinking it's a rifle, regardless of the pose of the object!



■ classified as turtle ■ classified as rifle
■ classified as other

Figure 1. Randomly sampled poses of a 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint². An unperturbed model is classified correctly as a turtle nearly 100% of the time.

(Credits: Athalye, Engstrom, Ilyas, Kwok (2018))

Neural networks are still very brittle!

Outline

① Representation Learning

Hierarchical Compositionality

Distributed Representations

Auto-Encoders

Word Embeddings

② Convolutional Neural Networks

③ Visualizing Representations

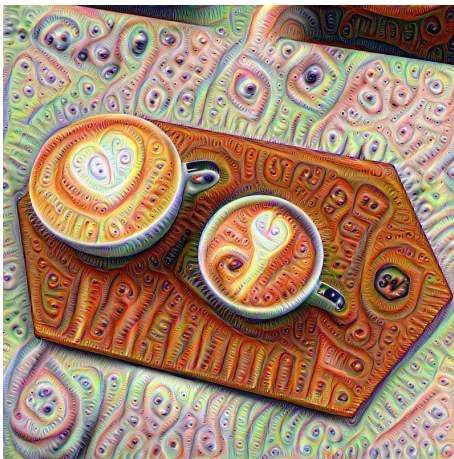
④ Conclusions

Conclusions

- Neural nets learn internal representations that can be transferred across tasks
- Distributed representations are exponentially more compact and allow generalizing to unseen objects
- Deeper neural nets exhibit hierarchical compositionality: upper level layers learn more abstract/semantic representations than bottom level layers
- Auto-encoders are an effective means for learning representations
- Word embeddings are continuous representations of words that are extremely useful in NLP
- Convolutional nets are extremely useful to capture translational invariances in images

Thank you!

Questions?



References I

- Artetxe, M., Labaka, G., and Agirre, E. (2017). Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 451–462.
- Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.
- Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.
- Cireřan, D., Meier, U., Masci, J., and Schmidhuber, J. (2011). A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE.
- Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2011a). Convolutional neural network committees for handwritten character classification. In *2011 International Conference on Document Analysis and Recognition*, pages 1135–1139. IEEE.
- Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011b). Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.
- Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.
- Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471.
- Ferreira, D., Almeida, M. S. C., and Martins, A. F. T. (2016). Jointly Learning to Embed and Predict with Multiple Languages. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.
- Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

References II

- Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.
- Hermann, K. M. and Blunsom, P. (2014). Multilingual Models for Compositional Distributional Semantics. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.
- Hinton, G. E. (1984). Distributed representations.
- Hubel, D. H. and Wiesel, T. N. (1965). Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of neurophysiology*, 28(2):229–289.
- Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- Lample, G., Ott, M., Conneau, A., Denoyer, L., and Ranzato, M. (2018). Phrase-based & neural unsupervised machine translation. *arXiv preprint arXiv:1804.07755*.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.
- Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proc. of the International Conference on Machine Learning*, pages 1058–1066.
- Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.